GER-3660

Digital Relay Software Quality

Digital Relay Software Quality

Charles R. Heising
Associated Power Analysts, Inc.
Philadelphia, PA

Ronald C. Patterson
Elaine Y. Weintraub
GE – Protection & Control
Meter & Control Business Dept.
Malvern, PA 19355

BACKGROUND

The first solid state electronic protective relays were introduced in the electric power industry in the late 1950's. The development of these relays using discrete components grew during the 1960's because of the potential for better accuracy, speed, and overall improvements in performance. Because of the number of parts used and the associated connections, as well as environmental sensitivity, the reliability of these electronic relays was not as good as the reliability of the equivalent electromechanical relays. This limitation has been partially offset by conservatively derating the parts to lower stress levels and utilization of high reliability parts to achieve reduced failure rates.

The use of electronic relays expanded in the 1970's following the introduction of low level integration in electronic components. The use of integrated circuits reduces the number of parts and the associated connections, thus improving the reliability and also making it easier to combine functions that had previously been performed with separate electromechanical relays.

The development of higher levels of integration in the 1980's has permitted the inclusion of functions such as internal monitoring to detect abnormal conditions and prevent incorrect relay operations. These features have been incorporated in contemporary hybrid relay systems such as GE's MOD-10 TYS transmission line protection system which combines analog signal processing with digital logic and self-test. These modular electronic relay systems include all of the Overcurrent, Distance, Logic, and Input/ Output functions required for a primary line protection terminal, and are functionally equivalent to about 35 electromechanical relay units.

DEVELOPMENT OF MICROPROCESSORS AND THE USE OF SOFTWARE

The development of microprocessors and high speed memories have led to the rapid growth of personal computers during the decade of the 1980's. Microprocessors and high speed memories have also been used to develop digital protective relays for application in the electric power industry. An example of this is the GE Digital Line Protection (DLP) relay which employs high resolution direct waveform sampling of the input current and voltage signals, digital signal processing and multiple microprocessor hardware, to provide all of the protection functions of the TYS as well as fault location and data communications capability. A digital relay like the DLP has only about half as much hardware as the hybrid TYS relay, but it requires the use of an extensive amount of software.

The rapid evolution in electronic relays raises two important new issues for the protection engineer. First, the effective design life to technical obsolescence has been dramatically reduced. ***Fig 1*** shows that the design life expectancy has shrunk from over 30 years with traditional electromechanical technology to approximately 5 years with the present rapidly changing electronic technology. This means that the utility protection

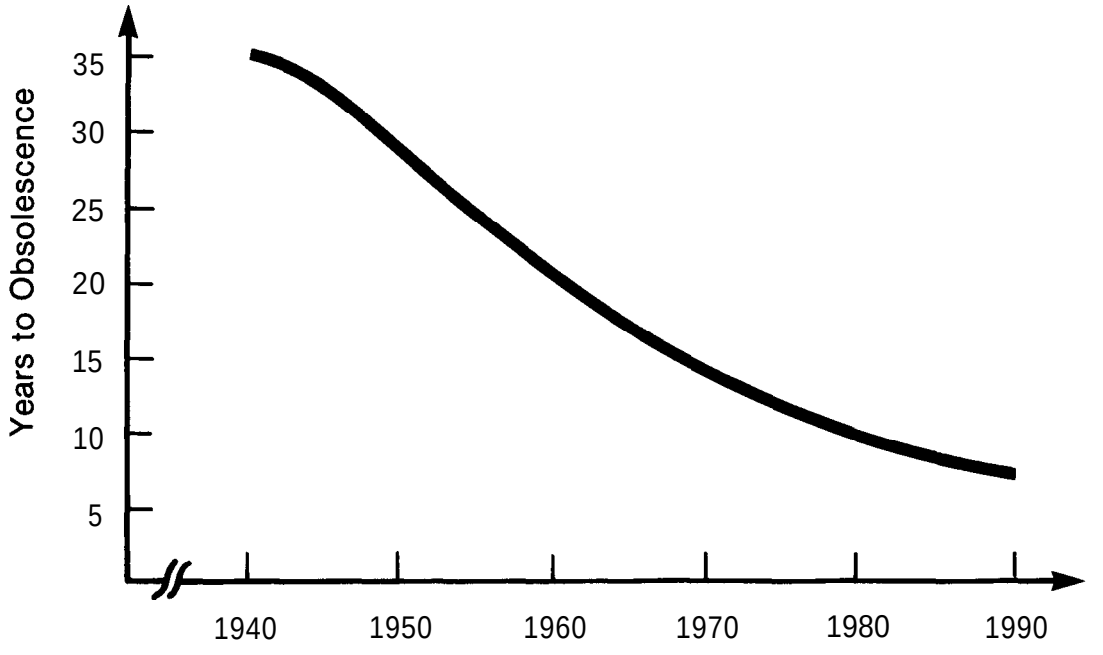

Figure 1
Protective Relay Design Life Expectancy

engineer can no longer make assessments of relay design reliability based on significant periods of field experience at trial installations before committing to widespread application across the power system.

The second issue is the question of the reliability of the software required for digital protective relay systems. ***Fig 2*** shows the changing hardware and software design content in electronic protective relays. For today's digital relays, only 20% of the total design effort involved is for hardware, compared to 80% for the software. Unfortunately, the analysis and prediction of software reliability is a much less mature discipline than that of the hardware reliability.

SOFTWARE RELIABILITY EXPERIENCE IN OTHER INDUSTRIES

What has been the experience with software problems in industries other than the electric power industry? What has been done to bring about significant improvements? What are some of the best practices that have been developed in order to bring about significant improvements in software reliability? What lessons can be learned from this and applied to the design of protective relays in the electric power industry?

Software reliability has been a major cause of in-service failures of both commercial systems and military systems. A recent example of this was the failure of about half of the AT&T long distance telephone system in the Northeastern part of the United States on January 15, 1990. This massive system failure was attributed to a software error in a switch. The AT&T system had self-monitoring on all of the switches; in the event of a failure, the switch was supposed to disconnect itself, allowing the traffic to be redirected to other circuits. The AT&T system should have kept operating after a switch failure because of extensive redundancy. Unfortunately, the software error resulted in the cascading "turn

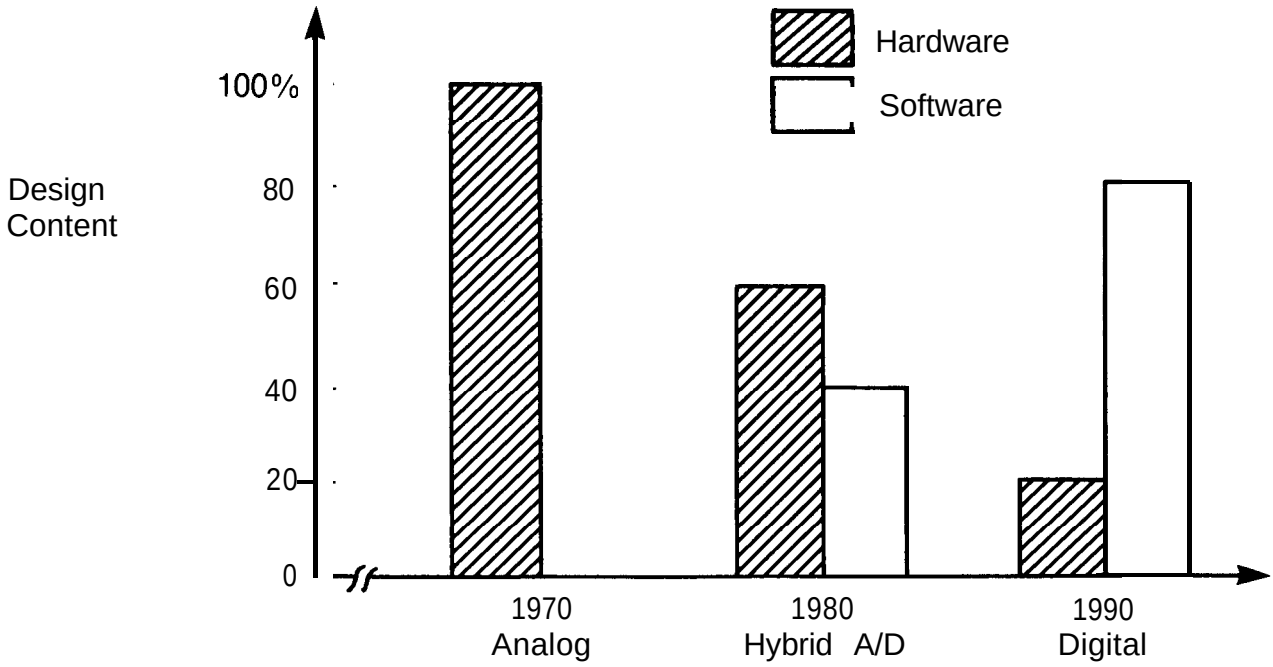

Figure 2
The Evolution of Electronic Relay Design (USA)

off" of many switches and defeated all of the redundancy and all of the internal self-monitoring that had been designed into the system. It was necessary for the Chairman of the Board of AT&T to go on national television and apologize to the public for what had happened and attempt to explain what the problem was. This incident is a good example of why it is important to give attention to the importance of software reliability.

DEFINITIONS OF SOFTWARE RELIABILITY AND QUALITY

The term "software reliability" usually refers to the probability of successfully performing the essential functions for a specific period of time. The term "software quality" usually refers to all of the software characteristics conforming to the specifications. In other words "software quality" includes many lesser errors that may not adversely affect the primary functions of the product. Typically the software errors that affect reliability may be only 20% of the total number of software errors that can effect quality.

DEVELOPMENT OF NATIONAL STANDARDS TO IMPROVE SOFTWARE QUALITY

During the decade of the 1980's software engineering personnel from several different industries got together and wrote a number of U. S. ANSI/IEEE standards on software quality [1] [2] [3] [4] [5] [6] [7] [8] [9]. These standards included software quality assurance plans, software verification and validation plans, software project management plans, software test plans and test documentation, software reviews and audits, etc. Many of the best practices that have been learned are included as "guideline standards."

One such standard is a formal procedure called "software inspection." "Software inspection" is a review process of the software design documents, the code, and the detailed test plans as they are generated during a development program. Studies have found that

this procedure results in a reduction in the number of software errors ranging from 60 % to 90% [10]. This is obviously an important procedure that should not be ignored. Other studies have concluded that even when this "software inspection" procedure is used the software still probably contains between 0.5 and 3.0 errors per thousand source lines of code [11].

HARDWARE AND SOFTWARE RELIABILITY / QUALITY FOR PROTECTIVE RELAYS

Digital relays such as the GE DLP have only about half as much hardware as their hybrid electronic counterparts such as the GE MOD-10 TYS. This results in correspondingly better hardware reliability. However, digital relays rely heavily on the software quality and this raises two very obvious questions: how reliable is the software? and what kind of failure modes can occur in field service due to software errors?

Hardware reliability measures that are often used for relays include both the failure rate and the percent of correct operations. Enough data have been collected on hardware-based electronic relays in field service so that reasonably accurate values for their failure rates are known [12]. It is also possible to predict the hardware failure rate of a new design using military handbooks such as MIL 217E [13]. These handbooks give component failure rates and show how they vary with percent of rated stress, ambient temperature, and other environmental factors. These handbook predictions can be adjusted by a correction factor in order to take into account actual failure rates of previous relay designs in field service.

A lot less is known about the software failure rates in field service and it is even more difficult to accurately predict the software reliability of a new design. There is a need to pay attention to the factors that can have a significant effect on the software failure rate. Several of these factors are discussed in the following sections.

SOFTWARE ENGINEERING

There is no doubt that the reliability of the software controlling the operation of a transmission line relay protection system is a critical element of its overall reliability. If that software ever fails to operate as expected, it matters very little that the electronic or electromechanical components are still functioning perfectly.

All failures of the software can be traced to design or implementation defects. Because of this, the answer to improving software reliability and quality lies in the reduction of the number of these defects. This means conformance with all explicitly stated product requirements. It also means conformance with all software design, coding and testing requirements. All product specifications must be clearly defined at the beginning. The software requirements must be determined before development starts, since these requirements drive the development process. A software quality assurance program must be carried out throughout the whole product development program in order to detect discrepancies in the conformance to requirements and defects in the software. The software quality assurance program consists of a series of reviews and tests. A detailed verification and validation program must be carried out at each step of the software design and coding process. It is important to detect and correct errors as early as possible in the software development program. The costs for making corrections increase substantially as the program progresses and are even higher after the product gets into field service at user installations. ***Fig 3*** shows the various software engineering activities involved and the relative effort involved in each area. Experience in other industries has shown that failure to adhere to appropriate software quality assurance measures in the development phase results in significant increases in the software maintenance area.

The best practices described in the ANSI/IEEE guideline standards for software quality are currently used in the GE software development programs. A recent example is the development of the Digital Line Protection (DLP) system.

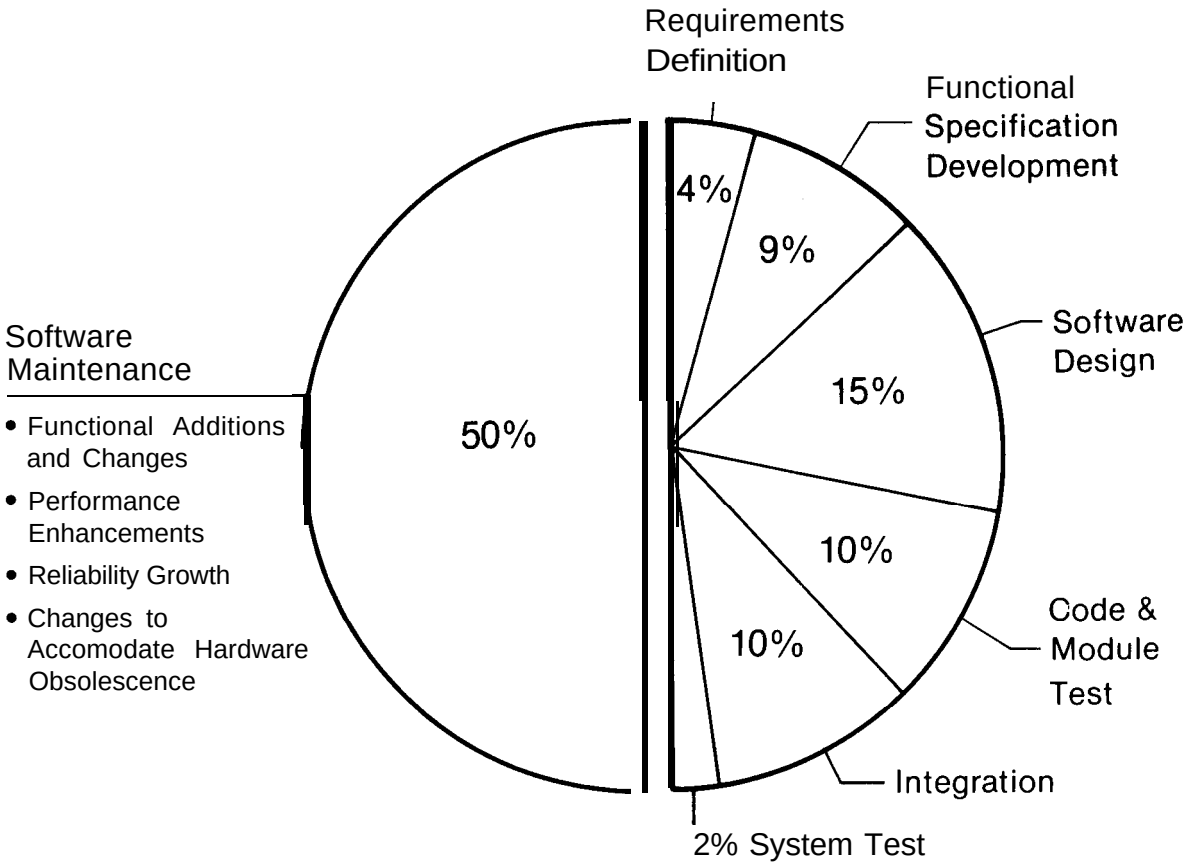

Figure 3
Typical Digital Relay Software Engineering Requirements Using Software Quality Assurance Program

SOFTWARE DEVELOPMENT PROGRAM FOR DLP

The DLP system uses mature readily available microprocessors, has finite memory size, must operate in real time, and must have the highest practical reliability. These requirements created software quality challenges because of: the need for a short design cycle; the high cost of software manpower for a quality product; and the need for reasonable product cost. ***Fig 4*** shows a diagram of the software development program up to the time of product release.

The first phase is always Requirements Definition. This document was primarily written by Marketing, Application Engineering, and Product Plannning with inputs from users. The writing of the requirements is an iterative process. Inputs from Software Engineering in the next phase of the project resulted in additional changes in the Requirements Definition in order to optimize the product requirements and software functional requirements.

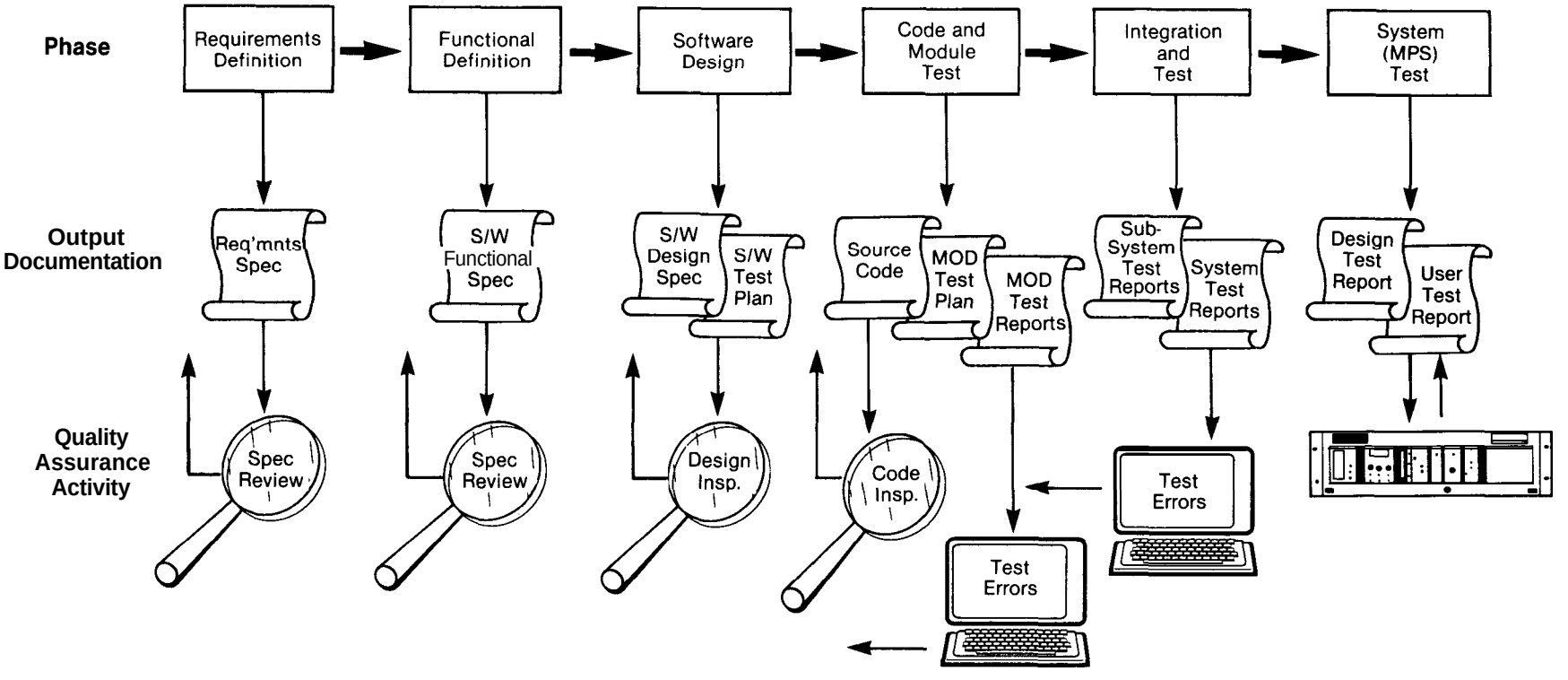

Figure 4
Software Engineering Life Cycle
Up to Product Release

The second phase was Functional Definition, which was a major Software Engineering activity. The overall product architecture was determined with both software and hardware engineers participating; this included the hardware/software partitioning. The DLP software was further partitioned into seven separate software subsystems. The specific breakdown was determined to be optimal partitioning to preserve crisp interfaces between subsystems, while still permitting rapid development through paralleling of the design tasks. Each of these seven software subsystems was a separate design package. The definition of self-test was a joint effort between the hardware and software design groups. The purpose of self-test is to detect hardware failures, and the hardware must be designed to permit self-test. Specific "software sanity checks" were also defined to detect hardware failures.

The Functional Definition phase also included rapid prototyping of certain software features that were being considered; such as, the man/machine interface and the remote communications. An evaluation was also made of potential algorithms. The software functional specification was developed over a period of nine months, and this became the basis for the detailed software work that took place during the remaining phases. The software functional specification was twelve times the size of the hardware functional specification.

The third phase was Software Design. The data connections between each of the software subsystems were defined. The processing and data flow requirements were written for each of the subsystems. Each of the seven subsystems was subdivided into separate logical modules. The number of modules per subsystem ranged from a low of five to a maximum of forty. Each module was a separate package that was assigned to a software engineer who then then did the detailed design. The methodology used for detailed design was "program design language" (PDL). The PDL is pseudo-English and describes all of the logical decision making that goes on in that module. The PDL designs for these more than one hundred modules were done in parallel in order to shorten the development time.

The fourth phase was Code and Module Test for each module. The detailed code was written for each module along with the plan for testing the code. The code was then tested using a VAX computer as an emulator.

The fifth phase was Integration and Test. The testing program includes an organized set of test plans. These test plans were intentionally not written by the module designer. The initial steps involved the merging of software modules together for testing and the application of software modules onto actual hardware. This was done in small steps and attempts were made to test as many interfaces and as many logic paths as possible. Ultimately a whole subsystem was tested together. The final step tested all the subsystems together in order to confirm the subsystem interfaces.

The sixth phase was System Test. This was done on the GE transmission line Simulator (Model Power System) and tested the overall product performance from CT and CVT inputs to circuit breaker trip and control outputs. The Model Power System provides full dynamic simulation of all types of faults in various locations for numerous primary power system configurations. The tests provide a comprehensive evaluation of relay performance under worst case combinations of variables which include load flow, fault resistance, incidence angle, CT saturation, CVT transients, mutual coupling and evolving faults. The relays and associated communications and control interact on a real-time basis with the Model Power System, tripping breakers to interrupt fault currents, and reclosing to introduce clearing and recovery transients. [14]

At the end of the System Test phase, when a product is put into production, the product is monitored at user installations in order to identify and correct any problems that are found.

The seventh phase is Maintenance. This includes: fixing any new errors that may be discovered after the start of production, adding enhancements to the product, adding new features to the product, and making any necessary software changes that may be caused by some of the hardware becoming obsolete. As can be seen from ***Fig 3***, this represents a very significant part of the total software engineering effort.

SOFTWARE QUALITY ASSURANCE PROGRAM

A software quality assurance program is carried out during each of the phases of the software development program. This program consists primarily of a series of disciplined reviews and a series of tests. Errors found during reviews are corrected before proceeding to the next phase. Errors found during tests are documented, corrected, and retested in order to prove that the error had been corrected. Retest is an important part of proving that the error was corrected by the fix, and that the fix did not introduce new errors.

Fig 4 shows the output documentation from each phase of a software development program. For the DLP, the total of all of this documentation was a stack of paper that weighed 150 pounds. ***Fig 4*** also shows the review and test activities that took place as part of the overall software quality assurance program. The software functional specification review, the design inspection, the code inspection, and the test procedure inspection are important elements of the program to detect both discrepancies in the conformance to requirements, and defects in the software before the test program is started. The complete test program was an organized effort that was part of the overall validation and verification program.

The “software functional specification review” for the DLP consisted of a series of formal reviews that included software engineers, hardware engineers, marketing and manufacturing representatives. These reviews consisted of page by page and paragraph by paragraph reviews. The software functional specification went through several revisions before being finalized. Any discrepancies or errors in this document can be both time consuming and costly to correct if not detected until later in the software development program or during field service at user installations.

“SOFTWARE INSPECTION” PROGRAM

The “design inspection” and “code inspection” steps on the DLP program were probably the most productive parts of the review process from a quality improvement standpoint. These “software inspections” were conducted in a formal manner with a team of approximately four to eight people participating in each review. The best practices for “software inspection” that have been developed in the computer industry have been documented in ANSI/IEEE 1028- 1988 [8].

Some examples of the types of defects uncovered during the DLP “software inspection” process include: missing functional capability, incorrect sequence of operations, invalid logical comparisons, and incomplete data structures.

On the DLP program a moderator was appointed for each “software inspection” who was not the author of the software element. The moderator was responsible for issuing a report after the inspection review and for following up to see that any necessary corrective actions were taken. A recorder was appointed to document the defects found at the meeting. The moderator functioned as a reader to lead the inspection team in a

comprehensive and logical fashion, reading line by line where required. Other team members functioned as inspectors to find defects in the software. The objectives of each "software inspection" were: verify that the software element satisfies its specification and conforms to applicable standards such as structured design/code constructs; uncover design flaws; document flaws and deviations from specifications and applicable standards; and, where appropriate, verify that the test procedure planned is adequate.

The appropriate documents to be reviewed were sent to each team member several days in advance. Each team member reviewed this information prior to the review meeting. At the meeting the entire team reviewed the software element, evaluating its condition relative to applicable specifications and standards. All flaws and deviations from specifications and standards were recorded. An exit decision was made for each software element reviewed: accept as is; rework and have moderator verify the rework; or reinspect. Follow up procedures were established to insure that no deviations were left uncorrected. In cases where reinspection was decided, the moderator scheduled another review. Each review was limited to two hour sessions at one time in order to avoid ineffectiveness due to fatigue.

DISCUSSION

The DLP "software inspection" effort is accompanied by a test program that starts with the modules, then builds into combinations of modules, followed by combinations of modules with their hardware, then a complete subsystem, and finally a complete system. The test program is as complete as can practically be accomplished. However, the entire test program does not guarantee that 100% of all software defects have been uncovered. It is not possible to test all combinations of internal software code functions that may occur in real time; indeed attempting to do so would require more than 10^{3000} individual tests.

Numerous self-test features have been designed into the DLP system. The main purpose of these features is to detect hardware failures so that they can be repaired before becoming an "incorrect relay operation." This permits a lengthening of the scheduled maintenance interval for the hardware and can be a significant cost saving to the user. It can only be estimated as to how effective these self-test features will be; a previous paper [12] gave a 75 % effectiveness estimate based upon field service experience of similar electronic relays. The DLP relay has been designed so that a hardware failure will not result in a false trip. When a hardware failure occurs, an alarm is activated and the tripping hardware is disabled. It has to be recognized that not all hardware failures can be detected by self-test, and there can also be failures in the self-test hardware.

The users of the DLP relay systems receive an instruction book that contains relay logic diagrams and tells how the relay system performs functionally. Users are not, however, expected to understand the details of the software, and are not expected to do reprogramming of the software. The software for the DLP relay has been burned into the memory and becomes "firmware." The "firmware" can only be changed by replacing the memory.

During the 1980's GE did microprocessor relay research and development work on an EPRI supported project [15]. Some of the present software development techniques and testing procedures were developed in connection with that project.

HARDWARE RELIABILITY PREDICTION FOR DLP

A failure rate prediction has been made for the hardware of the DLP relay system. This prediction used MIL Handbook 217E [13] for the component failure rates; and the overall failure rate was adjusted based upon previous experience obtained from comparing the field service reliability of relays with handbook predictions made during the past twenty years. The predicted hardware failure rate is 0.028 failures per year. It is further estimated that 75% of these failures can be detected by self-test. Thus the predicted operational failure rate is only one-fourth as much, or 0.007 failures per year. The self-test features significantly reduce the need for the user to perform scheduled maintenance to find hardware failures. This, in turn, permits a longer maintenance interval than that used for relays without the self-test features.

SOFTWARE/HARDWARE RELIABILITY PREDICTION

Current methodology does not permit an accurate relay reliability prediction that includes both the hardware and software. Field service reliability data have been collected in other industries [16]; and in many cases the number of software failures have exceeded the number of hardware failures. This has been a driving force in why software quality assurance programs have been developed and why attempts have been made to quantify how much improvement might be obtained during the development phase. Software quality assurance programs during the development phase offer the potential of substantially reducing the number of problems that are discovered after the product is placed in field service.

Most new products go through a period of reliability growth during the early years after their introduction into field service. Reliability growth can be defined as a reduction in the number of failures per unit time; this growth comes about from problems being uncovered and then eliminated by taking some appropriate form of corrective action. The patterns of hardware reliability growth have been well documented in the literature [16] and have been written into MIL Standards [17] and are in the process of being written into international standards by the International Electrotechnical Commission, Technical Committee No. 56 on Reliability & Maintainability. It is realistic to expect similar patterns of software reliability growth after the product is applied in service.

CONCLUSIONS

The evolution of digital technology in electronic protective relays involves major changes in the factors affecting performance and reliability, as well as the life expectancy to design obsolescence. Traditional methods of assessing relays by hardware inspection and testing are no longer adequate, since up to 80% of the engineering design content of contemporary digital relays is in the software area. It has therefore become increasingly important for the relay engineer to understand the requirements for assuring optimum relay software quality. The DLP digital line protection relay development described illustrates these essential software engineering design practices and their impact on complex systems. The maintainability as well as the reliability and performance of digital relays are highly dependent on the implementation of the software engineering design practices described and should be carefully considered by the protection engineer.

REFERENCES

1. **ANSI/IEEE Standard 730-1984, *"Software Quality Assurance Plans."***
2. **ANSI/IEEE Standard 829-1983, *"Software Test Documentation."***
3. **ANSI/IEEE Standard 982.1-1988, *"Standard Dictionary of Measures to Produce Reliable Software. ''***
4. **ANSI/IEEE Standard 982.20-1988, *"Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software"* (IEEE Standard 982.1-1988).**
5. **ANSI/IEEE Standard 983-1986, *"Software Quality Assurance Planning. ''***
6. **ANSI/IEEE Standard 1008-1987, *"Software' Unit Testing."***
7. **ANSI/IEEE Standard 1012-1986, *'Software Verification and Validation Plans.***
8. **ANSI/IEEE Standard 1028-1988, *"Software Reviews and Audits.***
9. **ANSI/IEEE Standard 1058.1-1987 *"Standard for Software Project Management Plans."***
10. **Michael E. Fagan, *"Advances in Software Inspections,"* IEEE Transactions on Software Engineering, Vol. SE-12, No. 7, July 1986, pp 744-751.**
11. **Ware Meyers, *"Can Software for Strategic Defense Initiative Be Error Free?"*, *"Computer"* magazine, November 1986, pp 61-67 (Bell Labs Study, T. R. Thomsen).**
12. **C. R. Heising, R. C. Patterson, *"Reliability Expectations for Protective Relays,"* Georgia Tech Relay Conference, May 4-6, 1988, Atlanta, GA.**
13. **MIL-HDBK-2 17E, October 27, 1986, *"Military Handbook, Reliability Prediction of Electronic Equipment."***
14. ***"General Electric Model Power System"* (GE publication GER-3225).**
15. **EPRI Research Project RP- 1359.**
16. **Herbert Hecht & Myron Hecht, *"Software Reliability in the System Context,"* IEEE Transactions on Software Engineering, Vol. SE- 12, No. 1, January 1986, pp 5l-58.**
17. **J.T. Duane, *"Learning Curve Approach to Reliability Monitoring,"* IEEE Transactions on Aerospace, Vol. AS-2/No. 2, April 1964, pp 563-566.**
18. **MIL-HDBK-189, *"Reliability Growth Management,"* February 1981.**